PAPER

Dynamic Macro-Based Heuristic Planning through Action Relationship Analysis

Zhuo JIANG[†], **Junhao WEN**[††a)], **Jun ZENG**[††], **Yihao ZHANG**[†], **Xibin WANG**[†], ***Nonmembers***, ***and*** **Sachio HIROKAWA**[†††], ***Member***

SUMMARY The success of heuristic search in AI planning largely depends on the design of the heuristic. On the other hand, previous experience contains potential domain information that can assist the planning process. In this context, we have studied dynamic macro-based heuristic planning through action relationship analysis. We present an approach for analyzing the action relationship and design an algorithm that learns macros in solved cases. We then propose a dynamic macro-based heuristic that appropriately reuses the macros rather than immediately assigning them to domains. The above ideas are incorporated into a working planning system called Dynamic Macro-based Fast Forward planner. Finally, we evaluate our method in a series of experiments. Our method effectively optimizes planning since it reduces the result length by an average of 10% relative to the FF, in a time-economic manner. The efficiency is especially improved when invoking an action consumes time.

key words: *AI planning, heuristic planning, macro, action relationship, dynamic heuristic*

1. Introduction

Searches in automated planning can be guided by extracting heuristics from declarative problem representations [1]. This approach has been commonly and successfully applied to classical domain-independent planning [2], [3]. In the idealized h^+ heuristic [4], the heuristic estimate for state s is the cost of an optimal s-plan in corresponding relaxed tasks, which is an admissible heuristic but NP-hard to compute [5], [6]. The Fast-Forward (abbreviated as FF) planning system [7] introduces the heuristic h^{FF}, which accounts for positive interactions between facts, and which achieved success at the AIPS-2000 planning competition. Many current heuristic functions seek an approximation to the optimal relaxation heuristic h^+. Several recent landmark-related heuristics [3], [8]–[11] have proved to be superior to additive h^{max} heuristics [3], [12].

In AI planning, successful heuristic searching requires appropriate design of the heuristic h. However, since planners must obtain an estimate for every node in the search process, computing the heuristic is time intensive. The total planning time rapidly increases with problem size because the number of expanded states in the planning problem grows exponentially [13]. Although most planners focus on domain-independent planning, each planning problem derives from a particular domain. If domain knowledge is lacking, the heuristic function may be misleading and inadequate. To overcome this problem and improve planner performance, researchers have developed techniques that extract domain knowledge from the problem itself or from solved plans.

Macro operator (*macro* for short), a sequence of original operators can be added as a single unit to a domain, is a form of domain knowledge [14] and the use of macros has been widely explored. Usually, macros are abstracted from solved plans based on frequency, furthermore, Macro-FF [15] learned macros through building abstract components, and Adrien [16] presented a method to obtain useful macros by statistical and heuristic filtering of a domain specific macro library. In fact, approaches based on operator relationship analysis [8], [17]–[19] can mine a deeper level of domain-dependent control knowledge. The landmark is based on analyzing literals and abstracting information on particular domains, but without knowing the dependence relationships among the actions. Chrpa [18], [19] investigated action dependency by analyzing its negative and positive effects and generated macros based on that. It was proved that swapping a pair of adjacent and independent actions still preserved the validity, and the new action sequence π' still can solve the original planning problem. The authors then extended their theory to planning optimization [19] but did not provide a generalized formulation.

After generated, macros should be used to speed up the future problems. The use of macros can be thought of as extending the neighborhood of a series of successors visible from each state to selectively introduce states which hitherto would only have been visible after the application of several steps [20]. Directly adding macros to the domain as input of planners is an efficient method to enhance the domain, and can be handled by any classic planner without modification. However, if the additional macros are chosen poorly, the performance of the planner will decrease due to the increased branching factor.

With analyzing the relationship between actions and generating macros, we developed a dynamic heuristic based on macros. The four contributions of our paper are detailed below.

Manuscript received May 22, 2014.
Manuscript revised September 2, 2014.
Manuscript publicized October 23, 2014.

[†]The authors are with the College of Computer Science, Chongqing University, Chongqing, 400044, China.
[††]The authors are with the School of Software Engineering, Chongqing University, Chongqing, 400044, China.
[†††]The author is with Research Institute for Information Technology, Kyushu University, Fukuoka-shi, 812–8581 Japan.
a) E-mail: wjhcqu@cqu.edu.cn

DOI: 10.1587/transinf.2014EDP7170

First, we study and extend an approach that investigates relationships among actions, which is the basic theory for generating macros. Learning macros based on action relationship can dig deep information from previous experience.

Second, we propose an algorithm that generates macros from solved cases. This algorithm imposes a length threshold and frequency restriction to exclude unwanted results from the domain.

The next key contribution is that we establish a dynamic heuristic that reuses the generated macros, and integrate the heuristic and the above developments into a working planning system called the Dynamic Macro-based FF (abbreviated as DM-FF) system.

Finally, our proposed approach was subjected to extensive experimental evaluation. We demonstrated the practical ability of our method; in particular, its ability to optimize planning by reducing the length of the result in a time-economic manner.

The remainder of the paper is structured as follows. Section 2 provides essential background information. The first main part of our approach—deciding how to analyze the solved case and proposing the algorithm to generate macros based on the analysis—is presented in Sect. 3. Section 4 presents the dynamic macro-based heuristic and extends our idea to a state-of-the-art planner. Our method is experimentally evaluated in Sect. 5. The performance of our proposal, DM-FF, is compared with that of FF. Conclusions and ideas for future research are presented in Sect. 6.

2. Preliminaries

Generally, the aim of automated planning is to find a sequence of actions that transforms the initial state of the environment into a different state where the goals are met.

Definition 1 A *planning task* is a 4-tuple $\Pi = \langle V, O, I, G\rangle$, where

- V is a finite set of propositional *state* variables, in which each *state* is a set of facts that are true at the current instant;
- O is a finite set of *operators*; and
- I and G, with $I, G \subseteq V$, denote the initial and final states, respectively. The final state is the target state of the process.

In classical planning, an *operator* $op \in O$ is a triple $op(x_1, \dots, x_m) = (pre(o), add(o), del(o))$, where $x_1, \dots, x_m$ are all of the variable symbols that appear somewhere in op, $pre(o), add(o)$ and $del(o)$ are preconditions and effects of op.

During execution, operators should be substituted with ground actions, each of which is an instantiation with possible constants and is expressed as $op(c_1, \dots, c_m)$ without changing the name, where c_i belongs to the set of object constants C.

Correspondingly, an *action a*, which is any ground instance of operator op, is a triple with the set of elements $(pre(a), add(a), del(a))$. And $pre(a)$ is the set of predictions representing the conditions required for action a. $add(a)$ and $del(a)$ are the sets of facts that become true and false, respectively, after action a is executed. $pre(a) \subseteq s$ indicates that action a can be executed in state s. Following the execution, denoted by $a(s)$, the new state $s' = (s - del(a)) \cup add(a)$.

Definition 2 Given a planning task $\Pi = \langle V, O, I, G\rangle$, an ordered sequence of actions $\pi = \langle a_1, \dots, a_n\rangle$ is a *plan* for solving problem Π if and only if consecutive applications of $\pi = \langle a_1, \dots, a_n\rangle$ from the initial state leads to a state in which all goals are satisfied, i.e., $G \subseteq a_n(a_{n-1}, \dots, a_1(I))$.

3. Macro Learning through Analyzing Action Relationship

3.1 Action Relationship Analysis

Our planning algorithm analyzes the reusability of a certain fragment of action sequence, called as macros, from the solved planning cases. In this section, we introduce an approach for analyzing the action dependency relationship in solved cases, and extend it to more general situations and describe a sufficient condition for qualification as a macro.

The plan $\pi = \langle a_1, \dots, a_n\rangle$ always implies dependent and independent relationships between actions; that is, a precondition of an action a_t is rendered true only by executing a previous action a_{t-k} such that $t, k \in N$, $1 < t \le n$, $0 < k < t$. In this situation, a_t depends on a_{t-k}, and their positions in the plan cannot be exchanged. The dependence relation is formally defined as follows.

Definition 3 Let $\langle a_1, \dots, a_n\rangle$ be an ordered sequence of actions. Action a_j is *straightly dependent* on action a_i (denoted as $a_i \rightarrow a_j$) if and only if $0 < i < j \le n$, $(add(a_i) \cap pre(a_j)) \neq \emptyset$, and $(add(a_i) \cap pre(a_j)) \not\subset (\cup_{t=i+1}^{j-1} add(a_t))$.

Let $E(a_i, a_j) = (add(a_i) \cap pre(a_j)) \backslash \cup_{t=i+1}^{j-1} add(a_t)$. From Definition 3, it is clear that $E(a_i, a_j) \neq \emptyset$ if and only if $a_i \rightarrow a_j$. a_j is *straightly dependent* on action a_i; that is, a_i is the last action that makes $E(a_i, a_j)$ nonempty. Without loss of generality, action a_j is defined as *dependent* on the effect of action a_i (denoted as $a_i \rightarrow^* a_j$) if and only if $a_i \rightarrow a_j$ or a sequence of integers $k_1, \dots, k_l$ $(l \ge 1)$ exists such that $a_i \rightarrow a_{k_1}, a_{k_1} \rightarrow a_{k_2}, \dots, a_{k_l} \rightarrow a_j$. Obviously, the *dependent* relationship is transitive. The negation of this relation is given by $a_i \not\rightarrow^* a_j$.

For a more general description of the dependent relationship, we introduce two special actions, $a_0 = (\emptyset, \emptyset, I)$ and $a_{n+1} = (G, \emptyset, \emptyset)$.

Definition 4 Let $\langle a_1, \dots, a_n\rangle$ be an ordered sequence of actions. Actions a_i and a_j $(i < j)$ are *independent* of the effects (denoted as $a_i \not\leftrightarrow a_j$) if and only if $a_i \not\rightarrow^* a_j$, $pre(a_i) \cap del(a_j) = \emptyset$ and $add(a_j) \cap del(a_i) = \emptyset$.

Proposition 1 Let $\pi = \langle a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n\rangle$ be a plan in the planning problem Π and $a_i \not\leftrightarrow a_{i+1}$. Then plan $\pi' = \langle a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n\rangle$ also solves planning problem Π.

This proposition, proven in literature [18], states that if two actions are adjacent and independent of the effects, their positions in the plan can be swapped, and the new sequence of actions will solve the original planning problem.

To distinguish the primitive action in the domain from the assembly of actions, we presented the detailed definition of the atomic action and macro as follows.

Definition 5 An *atomic action* is an action a that is directly grounded from an original operator in O. Clearly, an atomic action cannot be decomposed.

Definition 6 A *macro* is composed of several atomic actions, and in some state, the predictions and effects of the macro are identical to those of its constituents. Formally, a macro is an assembly of actions $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$, denoted as $a_{i_1,i_2,\ldots,i_m}$ where

- $pre(a_{i_1,i_2}) = (pre(a_{i_1}) \cup pre(a_{i_2})) \backslash add(a_{i_1}), \ldots,$ $pre(a_{i_1,i_2,\ldots,i_m}) = (pre(a_{i_1,i_2,\ldots,i_{m-1}}) \cup pre(a_{i_m})) \backslash$ $add(a_{i_1,i_2,\ldots,i_{m-1}}),$
- $del(a_{i_1,i_2}) = (del(a_{i_1}) \cup del(a_{i_2})) \backslash add(a_{i_2}), \ldots,$ $del(a_{i_1,i_2,\ldots,i_m}) = (del(a_{i_1,i_2,\ldots,i_{m-1}}) \cup del(a_{i_m})) \backslash add(a_{i_m}),$
- $add(a_{i_1,i_2}) = (add(a_{i_1}) \cup add(a_{i_2})) \backslash del(a_{i_2}), \ldots,$ $add(a_{i_1,i_2,\ldots,i_m}) = (add(a_{i_1,i_2,\ldots,i_{m-1}}) \cup add(a_{i_m})) \backslash del(a_{i_m}).$

From Definition 4, it is clear that (1) each subsequence of $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$ can also be assembled as a macro and (2) if $a_{j_1,j_2,\ldots,j_k}$ is another macro in the same planning problem and $i_m = j_1$, then $a_{i_1,i_2,\ldots,i_m,j_2,\ldots,j_k}$ also be composed as a macro.

The next proposition states the conditions for creating a macro with $m = 2$.

Proposition 2 Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan that solves planning problem Π and $i < j$ be indexes of actions in π. Assume that the following conditions hold:

- for every k such that $i < k < j$, $a_i \not\leftrightarrow a_k \vee a_k \not\leftrightarrow a_j$;
- for every k and x such that $i < k < x < j$, $\neg(a_i \not\leftrightarrow a_k) \wedge a_i \not\leftrightarrow a_x$ implies $a_k \not\leftrightarrow a_x$;
- for every l and x such that $i < x < l < j$, $\neg(a_l \not\leftrightarrow a_j) \wedge a_x \not\leftrightarrow a_j$ implies $a_x \not\leftrightarrow a_l$.

Then there exists a plan $\pi' = \langle a_1, \ldots, a_{i,j}, \ldots, a_n \rangle$ that also solves planning problem Π.

Proof: Clearly, when actions a_i and a_j are adjacent, they can be assembled into $a_{i,j}$ without losing the validity of the plan. If the actions are not adjacent then we can move the actions immediately preceding a_i or following a_j until a_i and a_j become adjacent. The intermediate actions are shifted by repeating the following steps:

1. Let a_x be the action immediately following a_i in the current plan, such that $a_i \not\leftrightarrow a_x$. By Proposition 1, we can swap a_i and a_x.

2. Let a_y be the action immediately preceding a_j in the current plan, such that $a_y \not\leftrightarrow a_j$. By Proposition 1, we can swap a_y and a_j.

3. Let a_k be the action between a_i and a_j with the largest index k in the current plan, such that $\neg(a_i \not\leftrightarrow a_k)$. This action can be moved until it follows a_j by repeated application of Proposition 1 (specifically, action a_k can be swapped with its immediately succeeding action until it follows a_j).

4. Let a_l be the action between a_i and a_j with the smallest index l in the current plan, such that $\neg(a_l \not\leftrightarrow a_j)$. This action can be moved until it precedes a_i by repeated application of Proposition 1.

Without loss of generality, the actions in Proposition 2 can also be composed as a macro. The following proposition extends the above conditions to more general cases with $m \geq 2$.

Proposition 3 Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan that solves planning problem Π and let $A_{i_1,\ldots,i_m} = \langle a_{i_1}, \ldots, a_{i_m} \rangle$ be a subsequence of π. If actions a_{i_k} and $a_{i_{k+1}}$ satisfy the assumption of Proposition 2 for each $1 \leq k < m$, then a plan $\pi' = \langle a_1, \ldots, a_{i_1,\ldots,i_m}, \ldots, a_n \rangle$ exists that also solves planning problem Π.

Proof: Without loss of generality, we assume that $i_1 < i_2 < \ldots < i_m$. By Proposition 2, $a_{i_1}, \ldots, a_{i_m}$ can be assembled by repeatedly processing the steps in Proposition 2 with action pairs.

3.2 Generating Macros from Solved Cases

In Sect. 3.1, we defined the dependency relationship and macro, and introduced the conditions for generating a macro. This section describes search algorithms for learning the macros in solved cases.

Definition 4 states that we should first compute the directly dependent relationship $\rightarrow$. The directly dependent relationships among all actions in plan π are incorporated in a matrix R^D. $R^D(i, j) = 1$ if and only if $a_i \rightarrow a_j$, otherwise $R^D(i, j) = 0$. For each predicate p, $d(p)$ refers to the last action that makes the predicate true. Obviously, if $i \in \{d(p) | p \in pre(a_j)\}$, then $a_i \rightarrow a_j$ and $R^D(i, j) = 1$. R^D is computed by Algorithm 1.

The loop (step 3) starts at the second index because a_1 is true in the initial state I, and a_1 is not dependent on any other action. Step 7 ensures that $d(p)$ refers to the final action that created it.

The matrixes R^S and R^I are separately used to indicate the relationship of $\rightarrow^*$ and $\not\leftrightarrow$: $R^S(i, j) = 1$ if and only if $a_i \rightarrow^* a_j$, otherwise $R^S(i, j) = 0$, and $R^I(i, j) = 1$ if and only if $a_i \not\leftrightarrow a_j$, otherwise $R^I(i, j) = 0$. As the transitive closure, R^S is computed by the Warshall algorithm [21] as shown in Algorithm 2.

We can now identify pairs of actions that can be assembled according to Proposition 2, which provides the conditions and steps for assembling two actions. We also define a matrix R^A that holds the relationships of actions that can be

Algorithm 1 Return the directly dependent relationship matrix R^D

```
1: initialize $R^D := 0$
2: set $d(p) := 1$ for each $p \in add(a_1)$
3: for $i$:=2 to $n$ do
4:     for each $j \in \{d(p) | p \in pre(a_i)\}$ do
5:         set $R^D(j,i) := 1$
6:     end for
7:     set $d(p) := i$ for each $p \in add(a_i)$
8: end for
9: return $R^D$
```

Algorithm 2 Return the dependent relationship matrix R^S

```
1: initialize $R^S := R^D$
2: for $k := 1$ to $n$ do
3:     for $i := 1$ to $n$ do
4:         if $R^S(i,k) = 1$ then
5:             for $j := 1$ to $n$ do
6:                 set $R^S(i,j) := R^S(i,j) \vee R^S(k,j)$
7:             end for
8:         end if
9:     end for
10: end for
11: return $R^S$
```

Algorithm 3 Return the set of macros P

```
1: initialize $M := \{1,2,...,sizeof(V)\}$; $c := 1$; $s := \varnothing$; $P := \varnothing$;
2: FindPAAction($M, R^A, c$)
3:
4: Function FindPAAction($M, R^A, c$)
5:   if $M = \varnothing$ return $P$
6:   if $\sum_{i \in M} R^A(c,i) \geq 1$
7:     for each $i \in M$
8:       if $R^A(c,i) = 1$
9:         $s := s \bigcup \{c\}$
10:        if $\sum_{i \in M} R^A(c,i) = 1$  $M := M \setminus \{c\}$
11:        $c := i$
12:        FindPAAction($M, R^A, c$)
13:        break
14:      end if
15:    end for
16:  end if
17:  if $\sum_{i \in M} R^A(c,i) = 0$
18:    $s := s \bigcup \{c\}$;
19:    if $|s| > 1$  $P := P \bigcup s$
20:    $s := \varnothing$
21:    $M := M \setminus \{c\}$
22:    $c := \min(M)$
23:    FindPAAction($M, R^A, c$)
24:  end if
25: end Function
```

assembled: $R^A(i, j) = 1$ if and only if a_i and a_j can form a macro action $a_{i,j}$, otherwise $R^A(i, j) = 0$. The matrix R^A can be calculated in polynomial time. Note that two adjacent actions can always be assembled (Proposition 2). The next step searches for all macros in the successful plans.

Let $G = \langle V, E\rangle$ be a directed acyclic graph, where each node in V corresponds to an action in plan π and $(v_i, v_j) \in E$ if and only if a_i and a_j can be assembled as a macro. Then R^A is the adjacency matrix of graph G and learning all macros is equivalent to searching for all sub-graphs in G. The following Algorithm 3 returns the set of macros P.

In each iteration, there will be one element of set M being tested and then deleted until M becomes empty. In line 5, when $M = \varnothing$ does not hold, the process continues and jumps into the next iteration, and eventually, the set M will be emptied and the program will end with the returning P.

Note 1 When implementing the algorithm 3, the following points should be noted:

1. Every subsequence from elements of P whose length exceeds 1 is also a macro (by Definition 6). However, large-grained macros are unlikely to be reused and will reduce the efficiency of the planning algorithm if added to the domain. To improve Algorithm 3, we impose a length threshold on the assembled sequences, and filter out macros that are longer than the threshold l^*.

Appropriately choosing the threshold usually bases on the feedback of experiments [22]. If the consumption of extracting macros is close to or even exceed that of the planning process, the value of l^* should stop growing. We illustrate this principle through experiments in Sect. 5.

2. The improved search algorithm will exclude long subsequences, but will find many assembled sequences that occur occasionally and are unlikely to repeat in future problems. Therefore, we impose a frequency constraint by which successful candidates will eventually join the domain. Formally, the support rate of macro $a_{i_1,i_2,...,i_m}$ in the training set, denoted as $S(a_{i_1,i_2,...,i_m})$, should follow

$$S(a_{i_1,i_2,...,i_m}) = \frac{M}{N} \geq s^*,$$

where M refers to the number of plans that contain $a_{i_1}, \ldots, a_{i_m}$ in the training set, N represents the size of the training set, and s^* is the lower support threshold.

4. Dynamic Macro-Based Heuristic Planning

Heuristic search algorithms perform a forward search from an initial state to a goal state using a heuristic function that estimates the distance to the goal [4]. Given a planning task $\Pi = \langle V, O, I, G\rangle$, for each state s reached in a forward search, The Heuristic Search Planner (abbreviated as HSP) roughly estimates the solution length of the corresponding relaxed task $\Pi' = \langle V, O', I, G\rangle$ by computing the following weight values

$$g_s(f) = \begin{cases} 0 & \text{if } f \in s, \\ i & \text{if } [\min_{o \in O, f \in add(o)} \Sigma_{p \in pre(o)} g_s(p)] = i - 1, \\ \infty & \text{otherwise.} \end{cases}$$

HSP assumes that facts are independently acquired. The weight of a set of facts F is computed by the additive heuristic as follows

$$h(s) = \sum_{f \in F} g_s(f).$$

Heuristic planning based on macros can be naturally improved by directly adding the appropriate macros to the domain. Any planner can handle the enhanced domain without any change. In this method, a macro carries the same weight as an atomic action (i.e., 1). However, since a

Table 1 Settings and results of various training dataset.

No.	#depots	#distributors	#trucks	#pallets	#hoists	#crates	avg. length	avg. time(s)
TrainSet1	1	2	2	3	3	2~10	26.6	0.013
TrainSet2	3	3	2	6	6	5~15	51.6	0.31
TrainSet3	3	3	2	10	6	5~15	53.8	0.28
TrainSet4	4~8	4~8	4~8	10~20	8~15	8~20	70.89	2.678

* In TrainSet4, there were 2 problems that couldn't be solved with FF, and they were excluded from the other experiments

Table 2 Macros that present in every plan.

No.	Macro	frequency	total frequency
1	Lift[Para1, Para2, Para3, Para4]-Load[Para1, Para2, Para5, Para4]	20	113
2	Lift[Para1, Para2, Para3, Para4]-Load[Para1, Para2, Para5, Para4]-Drive[Para5, Para4, Para6]	20	55
3	Load[Para1, Para2, Para3, Para4]-Drive[Para3, Para4, Para5]	20	55
4	Drive[Para1, Para2, Para3]-Load[Para4, Para5, Para1, Para3]	20	58
5	Drive[Para1, Para2, Para3]-Lift[Para4, Para5, Para6, Para3]	20	75
6	Drive[Para1, Para2, Para3]-Lift[Para4, Para5, Para6, Para3]-Load[Para4, Para5, Para1, Para3]	20	52
7	Unload[Para1, Para2, Para3, Para4]-Drop[Para1, Para2, Para5, Para4]	20	103

macro executes in more than one time step, equally weighting macros and atomic actions will likely generate a non-optimal plan. As an example, consider the following short planning task. The initial state is empty, the goals are $\{G_1, G_2\}$, and four actions are implemented:

$$op_1 = (\varnothing, G_1, \varnothing),$$
$$op_2 = (\varnothing, G_2, \varnothing),$$
$$op_3 = (\varnothing, G_2, G_1),$$
$$op_{1,3} = (\varnothing, G_2, \varnothing).$$

$op_{1,3}$ is a macro assembled by actions of op_1 and op_3. If $op_{1,3}$ is weighted as 1, then macros (op_1, op_2) and $(op_1, op_{1,3})$ are both weighted as 2. Therefore, the planner selects both action sequences as optimal solutions. This example highlights the need to adjust the weights of macros.

Definition 7 The *dynamic weight* of an action a is defined as

$$w(a) = \begin{cases} 1 & \text{if } a \text{ is an atomic action,} \\ \alpha * len - 1 + \dfrac{1}{\log_M(freq + 1)} & \text{if } a \text{ is a macro.} \end{cases}$$

Here, len is the length of the macro operator, M is the number of occurrences of a in the training set (as previously mentioned), and $freq$ is the frequency with which action a is selected as part of the solution. The parameter α is the adjustment parameter, which satisfies $\alpha \in [2/len, 1]$. When α is small, the computing method allocates higher priority to macros. $freq$ is initialized as $freq := M$. For each successful selection of action a, $freq$ is incremented by 1. Obviously, the weight of a macro a is less than len, and approaches $len - 1$ if continually selected as part of the solution.

The dynamic weight enables timely and effective selection of macros. Accordingly, the measure to estimate the dynamic difficulty involved in achieving from s is computed as following:

$$g_s^d(f) = \begin{cases} 0 & \text{if } f \in s, \\ \min_{o\in O, f\in add(o)}(w(o) + \Sigma_{p\in pre(o)} g_s^d(p)) & \\ & \text{if } \exists o \in O,\ f \in add(o), \\ \infty & \text{otherwise.} \end{cases}$$

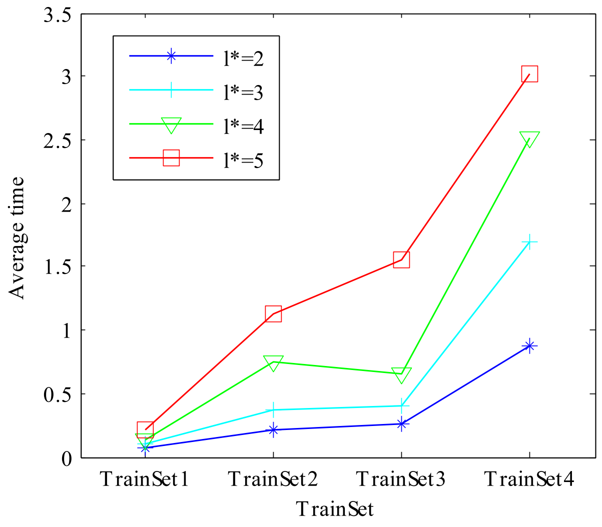

Fig. 1 Average execution time of varying length restriction.

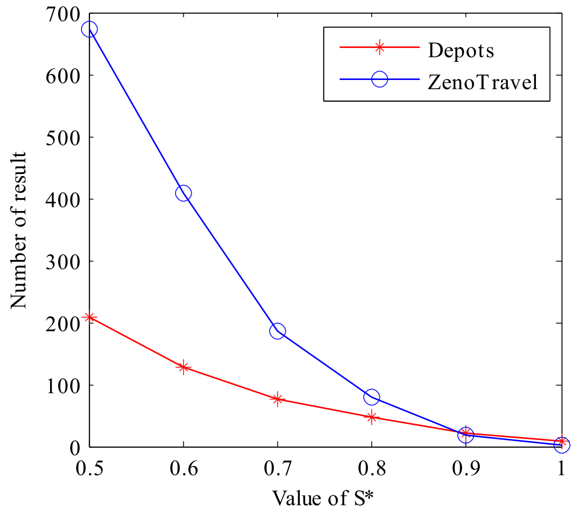

Fig. 2 Average number of valid result with varying frequency restriction.

The dynamic additive heuristic, based on generated macros, is then defined as

$$h^d(s) = \sum_{f\in F} g_s^d(f).$$

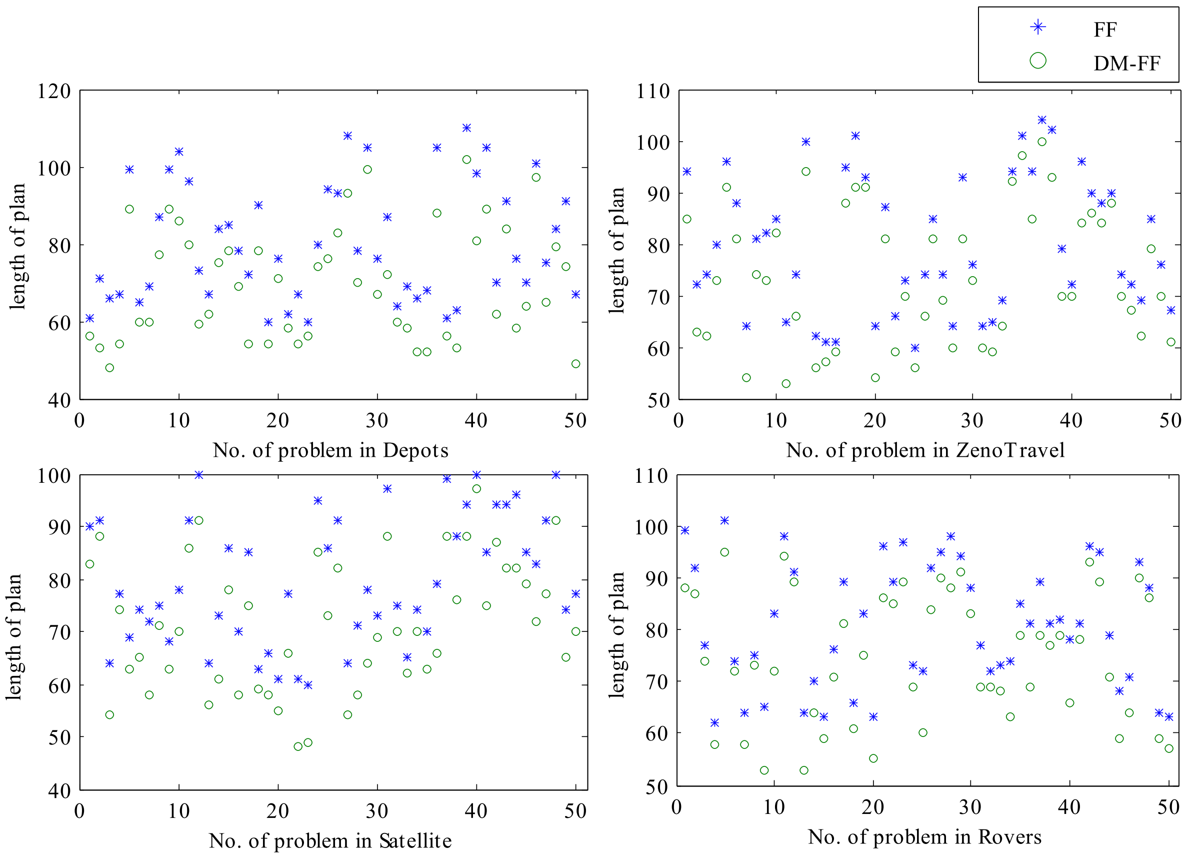

Fig. 3 Plan length comparison between proposed DM-FF and FF.

The above dynamic heuristic is also applicable to other heuristic-based planners. The FF planning system [7] is a state-of-the-art planner that can handle classical STRIPS planning problems. FF adopts a relaxed GRAPHLAN that estimates the number of time steps in the search and considers the positive effects among facts. As a case study, we indicate the growth of the time steps in FF by setting $w(a)$ instead of 1. The following section evaluates our improved FF system, DM-FF, in a series of experiments.

5. Performance Evaluation

We improved the macros generating algorithm according Note 1, called Learning Macros with Length and Frequency Restrictions (LMLF). The experimental training and test sets for each domain were generated by random generators supplied by the IPC2002 in the STRIPS version. The subjects were four representative domains: Depots, ZenoTravel, Satellite and Rovers. In the Depots domain, trucks transport crates around depots and distributors, and crates must be stacked onto pallets or atop other crates at their destination. In the ZenoTravel domain, aircraft transports travelers to their destination while monitoring fuel level. In the Satellite domain, one or more satellites conduct observations, collect data, and downlink the data to a ground station. In the Rovers domain, planning is implemented for several rovers equipped with different, but possibly overlapping, sets of equipment for traversing planet surfaces.

The first two experiments evaluated the effects of the thresholds l^* and s^* on the efficiency of the generating algorithm. The primary purpose of the Depots domain was to test STRIPS planners. In this domain, we generated 4 training datasets of varying difficulty, each consisting of 20 planning problems, and solved the planning problems using FF. The parameters of the random generators and the planning results are listed in Table 1.

This first experiment was mainly designed to reveal how the restriction l^* influences the execution time. However, according to the algorithms, it's clear that the plan length is a great factor and directly affects the performance. On the other hand, the plan length represents the complexity of the original planning problem. So we set up 4 training datasets with different complexities as shown in Table 1 and let the No. of the 4 datasets be the horizontal axis in Fig. 1.

Figure 1 shows the average time of searching for macros among the above planning results, varying the length restriction l^*. The total execution time, including the time required to generate the relationship matrices and search for the macros that meet the restrictions, was recorded and is displayed in this figure.

Execution time is influenced by plan length and generation of the relationship matrix. The latter consumes a large proportion of the total search time, especially in complex planning problems. As shown in Fig. 1, the search process

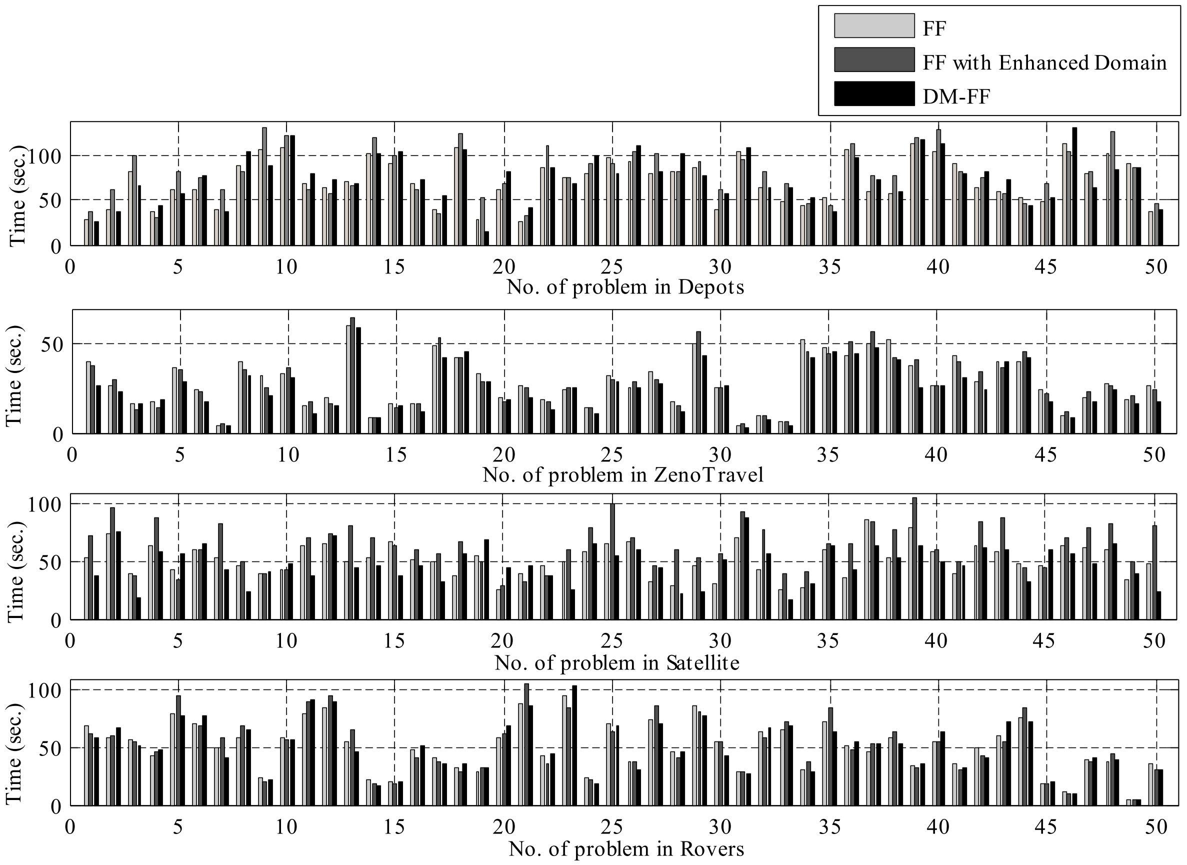

Fig. 4 Time comparison between different methods.

is more time-intensive under more relaxed conditions. In this experiment, when $l^* \geq 4$, the time of analyzing the results exceeds that of the planning process. Thus $l^* = 4$ is the maximum acceptable value of this threshold in this domain.

In another training dataset, we assess the effects of the threshold s^* on the number of valid results. The results for $l^* = 4$ are shown in Fig. 2. Clearly, the number of valid macros decreases as s^* increases.

From Fig. 2, we observe that in the Depots domain, even with s^* set to 1, seven macros are present in every plan. These macros are listed in Table 2.

In Table 2, the "frequency" of a macro refers to the number of plans containing the specified action in the training set, while "total frequency" monitors every appearance of this action. Note that a macro cannot be recognized solely by the action names contained within it. For example, in this experiment, the following macros both comprise "Lift" and "Load" actions, but they are clearly different because their preconditions and effects differ.

1. *Lift[Para1, Para2, Para3, Para4]-Load[Para1, Para2, Para5, Para4]*

2. *Lift[Para1, Para2, Para3, Para4]-Load[Para5, Para6, Para7, Para8]*

Finally, we added macros leading to $s^* = 1$ into corresponding domains and solved the problems respectively with FF and the proposed DM-FF. The quality of planning results, which means length of plan here, and the performance of our DM-FF comparing to FF were the examining objects of this experiment. For better illustration, the datasets were set much more complicated than the previous training datasets and each of them contained 50 planning problems. We considered the final weight of each macro action under the dynamic weighting scheme employed by the DM-FF planner as its contribution to the plan length.

The lengths of the planning results when problems in each domain are separately solved by FF and DM-FF are shown in Fig. 3. The horizontal axe indicates the number of each problem in the certain domain. In the four domains, a shorter plan can usually be found when using the proposed DM-FF, which suggests that the proposed dynamic heuristic is a practical method to optimize the planning. In fact, closer inspection of the detailed statistics reveals that DM-FF shortens the planning result by an average of 10.0% relative to FF. On the other hand, CPU computing time of each method was noted and compared in Fig. 4.

In addition to the original FF and proposed DM-FF, FF with enhanced domain, which means directly adding macros into the domain and keep the original FF planner, also has been compared in this experiment. Adding macros into domain will increase the solution space, and as shown in Fig. 4, sometimes FF with enhanced domain costs more computing time than the original FF. The proposed DM-FF, with improved dynamic heuristic, can get better performance than FF with enhanced domain. In fact, the results indicates that DM-FF is likely to improve the searching process and find a shorter plan in less time than the FF if the heuristic guides the planner to suitable macros.

6. Conclusions and Future Research

We have presented an approach for analyzing the action relationships in planning problems, from which we designed algorithms that search for available macros. To better exploit the potential domain information, we proposed a dynamic macro-based heuristic for guiding the planner toward more reasonable moves. As demonstrated by experiments, under appropriate restrictions, DM-FF can improve the searching process and find the plan in shorter time than the FF if the heuristic guides the planner to suitable macros. This feature improves the efficiency of the method, especially in situations where invoking an action is time consuming.

Macros can reduce the depth of the planning algorithm's search tree, but unavoidably, they enlarge the search tree's branching factor at the same time, which will decrease the benefits. If the original operators are too much and there is few intrinsic correlation between them, the inclusion of macros may damage the time and quality performance of the original action model. We have designed the dynamic heuristic to avoid this dilemma, but we still believe that there will more techniques can help to overcome the limitation in the future.

In addition, we hope to extend our idea to other fields, such as numeric domains and conformant planning problems. Potential action relationship may reasonably be considered to exist in all categories of planning tasks. We will also use the method to analyze dependency and other common relationships, with a view to improving the planning results.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to significantly improve the quality of this paper. The research reported in this paper has been partially supported by the National Science Foundations of China (Grant No. 61379158 and 71301177), the Ph.D. Programs Foundation of Ministry of Education of China (Grant No. 20120191110028), the Chongqing Science and Technology Commission, China (Grant No. cstc2013jcyjA40011).

References

[1] M. Ghallab, D. Nau, and P. Traverso, Automated planning: Theory & practice, Morgan Kaufmann, 2004.
[2] B. Bonet and H. Geffner, "Planning as heuristic search," Artificial Intelligence, vol.129, no.1-2, pp.5–33, 2001.
[3] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?," 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, pp.162–169, 2009.
[4] B. Bonnet and H. Geffner, "Hsp: Heuristic search planner," Ai Magazine, 1998.
[5] C. Betz and M. Helmert, "Planning with h+ in theory and practice," 32nd Annual German Conference on Artificial Intelligence, KI 2009, pp.9–16, 2009.
[6] T. Bylander, "Computational complexity of propositional strips planning," Artificial Intelligence, vol.69, no.1-2, pp.165–204, 1994.
[7] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," J. Artificial Intelligence Research, vol.14, pp.263–312, 2001.
[8] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," J. Artificial Intelligence Research, vol.22, pp.215–278, 2004.
[9] F. Pommerening and M. Helmert, "Optimal planning for delete-free tasks with incremental lm-cut," 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, pp.363–367, Sao Paulo, Brazil, 2012.
[10] E. Karpas and C. Domshlak, "Cost-optimal planning with landmarks," 21st International Joint Conference on Artificial Intelligence, IJCAI-09, July 2009, pp.1728–1733, International Joint Conferences on Artificial Intelligence, 2009.
[11] P. Haslum, J. Slaney, and S. Thiebaux, "Minimal landmarks for optimal delete-free planning," 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, pp.353–357, June 2012, Sao Paulo, Brazil, 2012.
[12] P. Haslum, B. Bonet, and H. Geffner, "New admissible heuristics for domain-independent planning," 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, AAAI-05/IAAI-05, pp.1163–1168, 2005.
[13] T. dela Rosa, A. Garcia-Olaya, and D. Borrajo, "A case-based approach to heuristic planning," pp.1–18, 2013.
[14] M.A.H. Newton, J. Levine, M. Fox, and D. Long, "Learning macro-actions for arbitrary planners and domains," Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007), pp.256–263, 2007.
[15] A. Botea, M. Enzenberger, M. Muller, and J. Schaeffer, "Macro-ff: Improving ai planning with automatically learned macro-operators," J. Artificial Intelligence Research, vol.24, pp.581–621, 2005.
[16] A. Dulac, D. Pellier, H. Fiorino, and D. Janiszek, "Learning useful macro-actions for planning with n-grams," 25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, pp.803–810, Nov. 2013.
[17] D. Cai, J. Sun, and M. Yin, "Conformant planning heuristics based on plan reuse in belief states," 23rd AAAI Conference on Artificial Intelligence and the 20th Innovative Applications of Artificial Intelligence Conference, AAAI-08/IAAI-08, pp.1780–1781, 2008.
[18] L. Chrpa and R. Bartak, "Towards getting domain knowledge: Plans analysis through investigation of actions dependencies," 21st International Florida Artificial Intelligence Research Society Conference, FLAIRS-21, pp.531–536, 2008.
[19] L. Chrpa, T.L. McCluskey, and H. Osborne, "Optimizing plans through analysis of action dependencies and independencies," 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, pp.338–342, Sao Paulo, Brazil, 2012.
[20] A. Coles and A. Smith, "Marvin: A heuristic search planner with online macro-action learning," J. Artificial Intelligence Research, vol.28, no.1, pp.119–156, 2007.
[21] S. Warshall, "A theorem on boolean matrices," J. ACM, vol.9, no.1, pp.11–12, 1962.
[22] S. Jimenez, T. De La Rosa, S. Fernandez, F. Fernandez, and D. Borrajo, "A review of machine learning for automated planning," Knowledge Engineering Review, vol.27, no.4, pp.433–467, 2012.

Zhuo Jiang received his B.S. degree from the Mathematics department of Xinjiang University, China, in 2008 and his MS degree in Computer Science from Chongqing University, China, in 2010. Currently, he is a Ph.D. candidate at Chongqing University, China. His research interest includes AI planning, Web service and data mining.

Junhao Wen received his B.S., M.S. and Ph.D. degrees in Computer Science from Chongqing University, China, in 1991, 1999 and 2008, respectively. Currently, he is a Professor and Ph.D. supervisor at Chongqing University, Chongqing, China.

Jun Zeng received his B.S. and M.S. degree in software engineering from Chongqing University in 2007 and 2010. He received Ph.D. degree in advanced information technology of Kyushu University in 2013. Since 2013, he has been a lecturer in the school of software engineering of Chongqing University. His research interest includes AI, data mining, and search engine.

Yihao Zhang received his B.S. degree and his M.S. degree in Computer Science from Xinyang Normal University and Kunming University of Science and Technology, China, in 2006 and 2010, respectively. Currently, he is a Ph.D. candidate at Chongqing University, China.

Xibin Wang is a Ph.D. student in School of Computer at Chongqing Uni-versity, China. He received his M.S. degree in computer science from Guizhou University in 2012. His research focuses on Computational Intelligence, data mining and business intelligence, and machine learning.

Sachio Hirokawa received the B.S. and M.S. degree in Mathematics and Ph.D. degree in Interdisciplinary Graduate School of Engineering Sciences from Kyushu University in 1977, 1979, and 1992. Since 1997, he has been a professor in research institute for information technology of Kyushu University. His research interest includes AI, search engine, text mining, and computational logic.